Protocol Solutions Group
3385 Scott Blvd.
Santa Clara, CA 95054

Net Protocol Suite User-Defined Decoding (UDD) Reference Manual

Manual Version 1.01

For Software Version 1.60

September 2014

Document Disclaimer

The information contained in this document has been carefully checked and is believed to be reliable. However, no responsibility can be assumed for inaccuracies that may not have been detected.

Teledyne LeCroy reserves the right to revise the information presented in this document without notice or penalty.

Trademarks and Servicemarks

Teledyne LeCroy is a trademark of Teledyne LeCroy.

Microsoft and *Windows* are registered trademarks of Microsoft Inc.

All other trademarks are property of their respective companies.

Copyright

© 2014 Teledyne LeCroy, Inc. All Rights reserved.

This document may be printed and reproduced without additional permission, but all copies should contain this copyright notice.

Table of Contents

Table of Contents iii
1 Introduction 1
2 Supported Scripting Languages 1
3 Architecture 1
3.1 Theory of Operation 1
3.2 Decoding Flow 1
4 Python API Reference 2
4.1 Modules 2
4.1.1 UDDEnum 2
4.1.1.1 Enumerations 2
4.2 Classes 2
4.2.1 OutContext 2
4.2.1.1 Members 2
4.2.2 Spec 3
4.2.2.1 Methods 3
4.2.3 InputContext 4
4.2.3.1 Methods 4
4.3 Globals 5
4.4 Entry Point Functions 5
5 Script Structure 6
6 Examples 6
How to Contact Teledyne LeCroy 7

1 Introduction

This manual describes how to define a script for decoding a particular protocol for Net Protocol Suite using User-Defined Decoding (UDD). You can define the format of protocols by introducing fields and their attributes: Name, Abbreviation, Description, MSB order and Byte Alignment. Additionally, you can decode a specific field by adding the options.

The scripts themselves are written in standard scripting languages and use a special Extension API to communicate with the Net Protocol Suite application.

2 Supported Scripting Languages

Python 2.7.x

3 Architecture

3.1 Theory of Operation

When a trace file is opened by the application, all frames inside the trace will first be decoded by the built-in decoder services. UDD allows you to extend and modify the built-in decoding services to your own definitions and concepts by creating custom decoding scripts, which have the file extension ".udd" and may be stored anywhere on your system. Once a script is created, it must then be assigned to a Protocol Type that exists in the trace file to be decoded. By doing this, the application communicates with your defined and assigned scripts for Protocol Types, initializes the fields and options, decodes the packets related to that specific protocol types, and change the packets structure in the GUI at the end.

Note: Refer to the Decoding Assignments section of the Net Protocol Suite User Manual for details on how to assign a script decoder for a protocol.

3.2 Decoding Flow

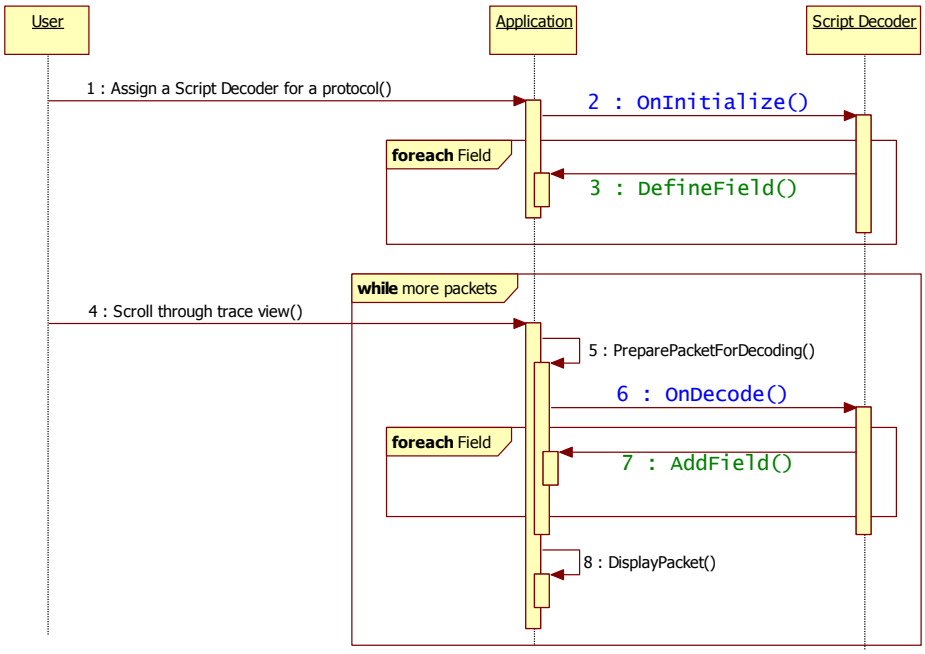

Figure 1. High-Level Script Decoding Sequence.

Figure 1 shows a high-level script decoding sequence. It shows that upon being assigned to a protocol in a trace file, a script decoder's `OnInitialize` function is called to define the set of fields (e.g., via `DefineField`) that will be handled by the script. Then, when a packet/frame from the trace file needs to be decoded, the application prepares the raw data and sends it to the script decoder through the `OnDecode` function, which will add the decoded fields (e.g., via `AddField`) to the packet/frame.

The entry point functions, `OnInitialize` and `OnDecode`, must be implemented in the script by the user. See section 4.4 for details on these functions.

4 Python API Reference

4.1 Modules

4.1.1 UDDEnum

In order to use the mandatory built-in enumerations, the script must import UDDEnum Python module.

4.1.1.1 Enumerations

BitOrder

This enumeration represents the Bit Order of a field.

Value	Description
MSB	Most-significant bit first.
LSB	Least-significant bit first.

ByteOrder

This enumeration represents the Byte order of a multi-byte field.

Value	Description
RIGHT_ALIGNED	Little endian.
LEFT_ALIGNED	Big endian.

4.2 Classes

4.2.1 OutContext

This class is a singleton and may not be instantiated in the script. The single instance is called “out”, which may be used by the OnDecode function to communicate special decoding results back to the application.

4.2.1.1 Members

`Result`

If this member is set to 1 by the OnDecode function, then any decoded fields added will be kept; otherwise, any decoded fields will be rolled back.

4.2.2 Spec

This class represents the static field definitions of a protocol's message formats.

4.2.2.1 Methods

`DefineField(name, abbreviation, tooltip, bit_order, byte_order)`

Defines a new field or updates an existing field.

Parameter	Type	Default Value	Description
name	string		Name of the field to be defined.
abbreviation	string		Abbreviation of the field to be defined.
tooltip	string		Tooltip of the field to be defined.
bit_order	BitOrder	BitOrder.MSB	The Bit Order of the field to be defined.
byte_order	ByteOrder	ByteOrder. LEFT_ALIGNED	The Byte Order of the field to be defined.
return value	int	1 - success 0 - error	

`DefineFieldDecode(name, decode_value, length, decode_name, decode_description)`

Define a custom decoding for an existing field. This decoding may override the application's built-in decoding for a field.

Parameter	Type	Default Value	Description
name	string		Name of the field to be decoded.
decode_value	unsigned int		Value of the field to be decoded.
length	int		Length in bits of the field
decode_name	string		The new name of decoded field.
decode_description	string		The description of decoded field.
return value	int	1 - success 0 - error	

`RenameField(name, new_name)`

Change an existing field name to a new name.

Parameter	Type	Default Value	Description
name	string		Name of the field to rename.
new_name	string		New name of the field.
return value	int	1 - success 0 - error	

4.2.3 InputContext

This class represents the packet/frame to be decoded. It provides access to the raw data, and it is the container for any decoded fields that are added.

4.2.3.1 Methods

`AddField(name, length, parent_id)`

Adds a field to the current packet being decoded.

Parameter	Type	Default Value	Description
name	string		Name of the field to rename.
length	int		The length in bits of the field.
parent_id	list	null	The identifier of Parent's field. If 'null', then the field has no parent.
return value	list	If a field has been successfully added to a Packet, its identifier will be returned as the return value.	

`GetFieldData(field_id, start_pos, length)`

Retrieves data from a field.

Parameter	Type	Default Value	Description
field_id	list		The field identifier.
start_pos	unsigned int		The start position value inside a field.
length	int		The data length in bits to retrieve from the field.
return value	list	A buffer list of Bytes retrieved data from a field.	

`GetData(start_pos, length)`

Retrieves data from a raw packet.

Parameter	Type	Default Value	Description
start_pos	unsigned int		The start position value in the packet.
length	int		The data length in bits to retrieve from the field.
return value	list	A buffer list of Bytes retrieved data from a field.	

4.3 Globals

`OutContext out`

The "`out`" singleton needs to be used inside the `OnDecode` function. If you confirmed the changes of decoding in the script, then `out.Result` must set to 1; otherwise, the decoding will not be applied.

4.4 Entry Point Functions

`OnInitialize(spec)`

This function must be defined in the decoding script. It allows you to initialize, define, and rename the fields for an existing built-in decoder or a custom script decoder. The application calls this function to initialize the `Spec` object that represents the protocol decoded by the script. In this function, the script must define the fields handled by the script by adding them to the `Spec` object. If you make any modifications on the fields definitions for a defined and assigned protocol type in the trace, then application just calls this function once to reorganize the structure of the fields. This function has one parameter named "`spec`" which is the interface to communicate with application for initialization of the user-defined fields' declarations and definitions.

Parameter	Type	Default Value	Description
spec	Spec		The Spec object that represents the protocol decoded by the script.
return value	none		

`OnDecode(incontext)`

This function must be defined in the decoding script. The application calls this function for each packet to be decoded. This function provides one parameter named "`incontext`" as an interface between application and your script. This parameter allows you to use your already defined fields from the `OnInitialize` function, add them to the packet and fetch the data from a field or packet in order to support your decoding procedure.

If you set "`out.Result = 1`" and then return from the function, then the decoded fields will be applied; otherwise, they will be rolled back.

Parameter	Type	Default Value	Description
incontext	InputContext		The Spec object that represents the protocol decoded by the script.
return value	none		

5 Script Structure

```
from UDDEnum import *

def OnInitialize (spec):
    # Initialize the Fields


def OnDecode (incontext):
    # Do the Decoding
    out.Result = 1
```

6 Examples

```
from UDDEnum import *

def OnInitialize(spec):
   spec.DefineField("Field", "Field Abbrev", "Field Tip", BitOrder.MSB, ByteOrder.LEFT_ALIGNED)
   spec.DefineField("Field 1", "Field1 Abbreviation", "Field1 Tooltip")
   spec.DefineField("Field 3", "Field3 Abbreviation", "Field3 Tooltip")
   spec.DefineField("Field 4", "Field4 Abbreviation", "Field4 Tooltip")
   spec.DefineField("Field 5", "Field5 Abbreviation", "Field5 Tooltip")
   spec.DefineField("Field 6", "Field6 Abbreviation", "Field6 Tooltip")

   spec.RenameField("Field 1","Field 2")
   spec.RenameField("Field 2","Field 1")
   spec.DefineFieldDecode("Field 3",0x04D2,16 ,"Option 1", "Option1 Description")
   spec.DefineFieldDecode("Field 3",0x05D2,16 ,"Option 2", "Option2 Description")

def OnDecode(incontext):
   fieldId =  incontext.AddField("Field", 32)
   fieldId1 = incontext.AddField("Field 1", 32)
   fieldId2 = incontext.AddField("Field 3", 16)
   fieldId3 = incontext.AddField("Field 4", 8,fieldId2)
   fieldId4 = incontext.AddField("Field 5", 8,fieldId2)
   incontext.AddField("Field 6", 32)

   fieldData = incontext.GetFieldData(fieldId1,0,32)
   fieldbufferlength = len(fieldData)
   fieldoutfile = open("c:/temp/FieldFilePython", "wb")
   for i in range(0,fieldbufferlength):
      fieldoutfile.write("%x\n" % fieldData[i])
   fieldoutfile.close()


   if len(fieldData) != 0 :
      if fieldData[0] == 0x28 :
         frameData = incontext.GetData(64, 64)
         framebufferLength = len(frameData)
         outFile = open("c:/temp/FrameFilePython", "wb")
         for i in range(0,framebufferLength):
            outFile.write("%x\n" % frameData[i])
         outFile.close()

   out.Result = 1
```

How to Contact Teledyne LeCroy

Type of Service	**Contract**
Call for technical support…	US and Canada: 1 (800) 909-2282 Worldwide: 1 (408) 727-6600
Fax your questions…	Worldwide: 1 (408) 727-6622
Write a letter …	Teledyne LeCroy Customer Support 3385 Scott Blvd. Santa Clara, CA 95054
Send e-mail…	support@teledynelecroy.com
Visit Teledyne LeCroy web site…	teledynelecroy.com/